

EQUAZIONI DI RICORRENZA

PIETRO DI LENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
UNIVERSITÀ DI BOLOGNA

ALGORITMI E STRUTTURE DI DATI
ANNO ACCADEMICO 2021/2022



INTRODUZIONE

- Una **equazione di ricorrenza** descrive ogni elemento in una sequenza in termini degli elementi precedenti

- Abbiamo già visto l'equazione di ricorrenza di Fibonacci

$$T(n) = \begin{cases} 1 & n \leq 2 \\ T(n-1) + T(n-2) + 1 & n > 2 \end{cases}$$

- Vogliamo determinare l'ordine di crescita delle equazioni di ricorrenza

- Determinare la crescita asintotica degli **algoritmi ricorsivi**

- Vedremo quattro metodi per risolvere equazioni di ricorrenza

- Metodo dell'iterazione
 - Metodo della sostituzione
 - Metodo dell'albero di ricorsione
 - Master Theorem

NOZIONI PRELIMINARI

- Le equazioni di ricorrenza di algoritmi ricorsivi possono essere del tipo

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(\lfloor n/3 \rfloor) + O(n) & n > 1 \end{cases}$$

- Per convenienza, tipicamente sostituiamo la notazione asintotica con **valori costanti**

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\lfloor n/3 \rfloor) + cn & n > 1 \end{cases}$$

- Il comportamento asintotico non è influenzato da **arrotondamenti**

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/3) + cn & n > 1 \end{cases}$$

METODO DELL'ITERAZIONE

- Il **metodo dell'iterazione** è un approccio di tipo *brute force*
- Idea: sostituiamo iterativamente la parte ricorsiva nell'equazione finché non appare uno **schema ricorsivo** legato al **passo di iterazione**
- Esempio. Consideriamo la ricorrenza $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + c & n > 1 \end{cases}$

$$T(n) = T(n/2) + c \quad \text{passo 1}$$

$$= T(n/4) + c + c \quad \text{passo 2}$$

$$= T(n/8) + c + c + c \quad \text{passo 3}$$

...

$$= T(n/2^i) + c \cdot i \quad \text{passo } i$$

La ricorsione termina quando $n/2^i = 1 \implies i = \log_2 n$. Quindi

$$T(n) = T(1) + c \cdot \log_2 n = 1 + c \cdot \log_2 n = \Theta(\log n)$$

METODO DELLA SOSTITUZIONE

- Il **metodo della sostituzione** può essere usato per *validare* un'ipotesi
- Idea: 1) **ipotizziamo** una soluzione 2) **validiamo** induttivamente l'ipotesi
- Esempio. Consideriamo la ricorrenza $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$

Ipotizziamo $T(n) = O(n)$, che implica

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \leq cn$$

- 1** Base. $T(1) = 1 \leq c \cdot 1$, per ogni $c \geq 1$
- 2** Induzione. Assumiamo che l'ipotesi sia vera per $T(n/2)$

$$\begin{aligned} T(n) &= T(n/2) + n \\ &\leq cn/2 + n && \text{(assumiamo } T(n/2) \leq cn/2) \\ &= (c/2 + 1)n && \text{(dobbiamo mostrare che } (c/2 + 1)n \leq cn) \end{aligned}$$

Il passo induttivo è vero se $\exists c > 0$ tale che $(c/2 + 1) \leq c \Rightarrow c \geq 2$

Concludiamo che **$T(n) = O(n)$** (vera $\forall c \geq 2$ e $n_0 = 1$)

METODO DELLA SOSTITUZIONE: FIBONACCI

- Cerchiamo un **limite superiore** alla ricorrenza di Fibonacci

$$T(n) = \begin{cases} 1 & n \leq 2 \\ T(n-1) + T(n-2) + 1 & n > 2 \end{cases}$$

Ipotizziamo $T(n) = O(2^n)$, che implica

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \leq c2^n$$

1 Base. $T(1) = 1 \leq c \cdot 2^1$, $T(2) = 1 \leq c \cdot 2^2$ vera $\forall c \geq 1/2 > 1/4$

2 Induzione. Assumiamo che l'ipotesi sia vera per $T(n-1), T(n-2)$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\leq c2^{n-1} + c2^{n-2} + 1 \\ &= c2^{n-2}(2 + 1) + 1 \\ &\leq c2^{n-2}(2 + 2) && (\text{vera } \forall n \geq 2 - \log_2 c) \\ &= c2^n \end{aligned}$$

Concludiamo che $T(n) = O(2^n)$ (vera $\forall c \geq 1/2, n_0 \geq 2 - \log_2 c$)

METODO DELLA SOSTITUZIONE: FIBONACCI

- Cerchiamo un **limite inferiore** alla ricorrenza di Fibonacci
- Ipotizziamo $T(n) = \Omega(2^{n/2})$, che implica

$$\exists c > 0 \text{ e } \exists n_0 \geq 0 \text{ tale che } \forall n \geq n_0, T(n) \geq c2^{n/2}$$

- 1 Base. $T(1) = 1 \geq c \cdot 2^{1/2}$, $T(2) = 1 \geq 2c$ vera per $0 < c \leq 1/\sqrt{2}$
- 2 Induzione. Assumiamo che l'ipotesi sia vera per $T(n-1)$, $T(n-2)$

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\geq c2^{\frac{n-1}{2}} + c2^{\frac{n-2}{2}} + 1 \\ &= c2^{\frac{n-2}{2}} \cdot (2^{\frac{1}{2}} + 1) + 1 && \left(2^{\frac{n-1}{2}} = 2^{\frac{n-2}{2}} \cdot 2^{\frac{1}{2}} \right) \\ &\geq c2^{\frac{n-2}{2}} \cdot (2^{\frac{1}{2}} + 1) \\ &\geq c2^{\frac{n-2}{2}} \cdot 2 && \left(2^{\frac{1}{2}} + 1 > 2 \right) \\ &= c2^{n/2} \end{aligned}$$

Concludiamo che $T(n) = \Omega(2^{n/2})$ (vera $\forall 0 < c \leq 1/\sqrt{2}$ e $n_0 = 0$)

LIMITE STRETTO PER LA RICORRENZA DI FIBONACCI

- Abbiamo dimostrato che la ricorrenza di Fibonacci è limitata da

$$T(n) = \Omega(\sqrt{2}^n) \approx \Omega(1.41^n) \text{ e } T(n) = O(2^n)$$

- Possiamo trovare un limite più stretto?

Teorema

Sia $T(n)$ l'equazione di ricorrenza di Fibonacci. Allora, $T(n) = 2F_n - 1$
(Dimostrazione per induzione)

- Dal Teorema sopra abbiamo che $T(n) = \Theta(F_n)$
- L' n -esimo numero di Fibonacci è definito da $F_n = \frac{1}{\sqrt{5}} \left(\phi^n - \hat{\phi}^n \right)$

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \text{ e } \hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$$

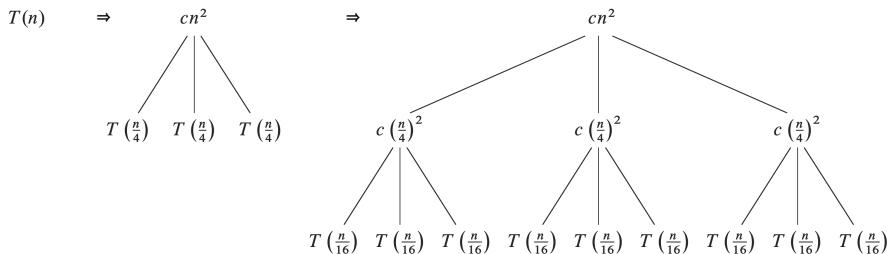
- Concludiamo che $T(n) = \Theta(\phi^n) \approx \Theta(1.62^n)$
- Difficile *indovinare* il valore ϕ^n con il metodo della sostituzione

METODO DELL'ALBERO DI RICORSIONE

- In un **albero di ricorsione** un nodo esprime il costo di un sottoproblema
- Idea: può essere visto come la *versione su albero* del metodo iterativo
 - 1 Generiamo l'albero di ricorsione dall'equazione di ricorrenza
 - 2 Calcoliamo il numero di nodi ad ogni livello dell'albero
 - 3 Identifichiamo qualche **schema ricorrente** legato al **livello dell'albero**
- Può essere complesso formulare un'ipotesi (metodo della sostituzione)
 - L'albero di ricorsione può essere usato per generare ipotesi
 - Tali ipotesi possono poi essere validate col metodo di sostituzione

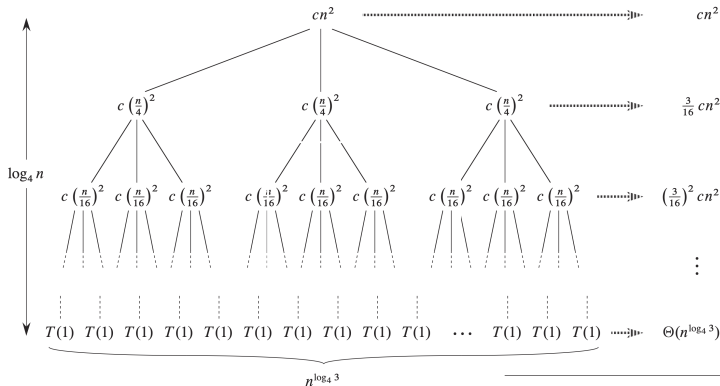
ESEMPIO 1: ALBERO DI RICORSIONE

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T(n/4) + cn^2 & n > 1 \end{cases}$$



- La radice rappresenta il costo al livello superiore della ricorsione
- I tre sottoalberi $T(n/4)$ rappresentano il costo su dimensione $n/4$
- Costo di $T(n/4)$: $c(n/4)^2 +$ costo di 3 chiamate ricorsive su $n/16$

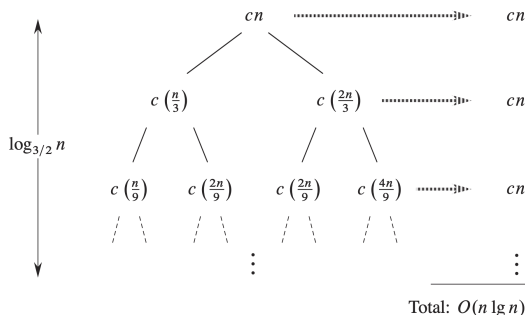
ESEMPIO 1: ALBERO DI RICORSIONE



- Se la radice è a livello 0, il costo a livello i è $\left(\frac{3}{16}\right)^i cn^2$ Total: $O(n^2)$
- La ricorsione termina quando $n/4^i = 1 \implies i = \log_4 n$
- $T(n) = \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i cn^2 \leq cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i = cn^2 \frac{1}{1-(3/16)} = O(n^2)$
- N.B. cn^2 è anche il costo della sola radice, quindi $T(n) = \Omega(n^2)$

ESEMPIO 2: ALBERO DI RICORSIONE

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/3) + T(2n/3) + cn & n > 1 \end{cases}$$



- Il percorso più **lungo** radice-foglia è

$$n \rightarrow \frac{2}{3}n \rightarrow \frac{4}{9}n \cdots \rightarrow \left(\frac{2}{3}\right)^i n \rightarrow \cdots \rightarrow 1$$

- Il percorso più **corto** radice-foglia è

$$n \rightarrow \frac{1}{3}n \rightarrow \frac{1}{9}n \rightarrow \cdots \rightarrow \left(\frac{1}{3}\right)^i n \rightarrow \cdots \rightarrow 1$$

ESEMPIO 2: ALBERO DI RICORSIONE

- La ricorsione sul percorso radice-foglia più **lungo** termina quando

$$\left(\frac{2}{3}\right)^i n = 1 \implies i = \log_{2/3} \frac{1}{n} = \frac{-\log_{3/2} n}{\log_{3/2} 2/3} = \log_{3/2} n$$

- **Upper bound** per $T(n) = T(n/3) + T(2n/3) + cn$

$$T(n) \leq \sum_{i=0}^{\log_{3/2} n} cn = cn \sum_{i=0}^{\log_{3/2} n} 1 = cn \log_{3/2} n = O(n \log n)$$

- La ricorsione sul percorso radice-foglia più **corto** termina quando

$$\left(\frac{1}{3}\right)^i n = 1 \implies i = \log_{1/3} \frac{1}{n} = \frac{-\log_3 n}{\log_3 1/3} = \log_3 n$$

- **Lower bound** per $T(n) = T(n/3) + T(2n/3) + cn$

$$T(n) \geq \sum_{i=0}^{\log_3 n} cn = cn \sum_{i=0}^{\log_3 n} 1 = cn \log_3 n = \Omega(n \log n)$$

- Conclusione: $T(n) = \Theta(n \log n)$

MASTER THEOREM

- Il **Master Theorem** è un approccio per risolvere ricorrenze della forma

$$T(n) = aT(n/b) + f(n)$$

con $a \geq 1$ e $b > 1$ **costanti** e $f(n)$ **asintoticamente positiva**

- Equazioni di ricorrenza di algoritmi che
 - dividono un problema di dimensione n in $a \geq 1$ sottoproblemi
 - tutti i sottoproblemi hanno dimensione n/b , con $b > 1$
 - il costo di ogni chiamata ricorsiva è dato da $f(n)$
- Non può essere applicato a tutte le possibili ricorrenze
 - Non può essere applicato all'equazione di ricorrenza di Fibonacci
 - Non può essere applicato all'esempio 2 (albero di ricorsione)

MATHER THEOREM

Theorem (Master Theorem)

Si consideri la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

dove $a \geq 1$, $b > 1$, d costante e $f(n)$ è una funzione di costo. Allora

- 1** *Se $f(n) = O(n^{\log_b a - \epsilon})$ per qualche $\epsilon > 0$ allora $T(n) = \Theta(n^{\log_b a})$*
- 2** *Se $f(n) = \Theta(n^{\log_b a})$ allora $T(n) = \Theta(n^{\log_b a} \log n)$*
- 3** *Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ per qualche $\epsilon > 0$, e se $a \cdot f(n/b) \leq c \cdot f(n)$ per qualche costante $c < 1$ e per tutti gli n sufficientemente grandi, allora $T(n) = \Theta(f(n))$*

N.B. In tutti e tre i casi confrontiamo $f(n)$ con $n^{\log_b a}$

MATHER THEOREM: VERSIONE SEMPLIFICATA

Theorem (Master Theorem)

Si consideri la seguente equazione di ricorrenza

$$T(n) = \begin{cases} d & n = 1 \\ aT(n/b) + cn^\beta & n > 1 \end{cases}$$

dove $a \geq 1$, $b > 1$ e c, d costanti. Sia $\alpha = \log_b a = \frac{\log a}{\log b}$. Allora

- 1** *Se $\alpha > \beta$ allora $T(n) = \Theta(n^\alpha)$*
- 2** *Se $\alpha = \beta$ allora $T(n) = \Theta(n^\alpha \log n)$*
- 3** *Se $\alpha < \beta$ allora $T(n) = \Theta(n^\beta)$*

N.B. Meno generale della versione precedente: ammette solo funzioni $f(n)$ della forma cn^β

ESEMPI: MASTER THEOREM

1 Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + c & n > 1 \end{cases}$

Abbiamo $a = 1, b = 2, \alpha = \log_b a = \log_2 1 = 0, \beta = 0$

$$\alpha = \beta \Rightarrow T(n) = \Theta(n^\alpha \log n) = \Theta(\log n)$$

2 Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & n > 1 \end{cases}$

Abbiamo $a = 1, b = 2, \alpha = \log_b a = \log_2 1 = 0, \beta = 1$

$$\alpha < \beta \Rightarrow T(n) = \Theta(n^\beta) = \Theta(n)$$

3 Risolvere col Master Theorem $T(n) = \begin{cases} 1 & n = 1 \\ 3T(n/4) + cn^2 & n > 1 \end{cases}$

Abbiamo $a = 3, b = 4, \alpha = \log_4 3 \approx 0.79, \beta = 2$

$$\alpha < \beta \Rightarrow T(n) = \Theta(n^\beta) = \Theta(n^2)$$

ESEMPIO: RICORRENZA DELLA RICERCA BINARIA

Cercare la posizione di un valore all'interno di un array ordinato (-1 se non trovato)

```
1: function SEARCH(ARRAY  $A[1 \cdots n]$ , INT  $x$ , INT  $i$ , INT  $j$ )  $\rightarrow$  INT
2:   if  $i > j$  then
3:     return  $-1$ 
4:   else
5:      $m = (i + j) / 2$   $\triangleright$  Divisione intera
6:     if  $A[m] == x$  then
7:       return  $m$ 
8:     else if  $A[m] > x$  then
9:       return SEARCH( $A, x, i, m - 1$ )
10:    else
11:      return SEARCH( $A, x, m + 1, j$ )
```

- La prima chiamata è invocata con parametri SEARCH($A, x, 1, n$)
- **Caso ottimo** (x è nel centro dell'array): $O(1)$
- **Caso pessimo** (x non è nell'array): ?

ESEMPIO: RICORRENZA DELLA RICERCA BINARIA

```
1: function SEARCH(ARRAY  $A[1 \cdots n]$ , INT  $x$ , INT  $i$ , INT  $j$ )  $\rightarrow$  INT
2:   if  $i > j$  then
3:     return  $-1$ 
4:   else
5:      $m = (i + j)/2$   $\triangleright$  Divisione intera
6:     if  $A[m] == x$  then
7:       return  $m$ 
8:     else if  $A[m] > x$  then
9:       return SEARCH( $A, x, i, m - 1$ )
10:    else
11:      return SEARCH( $A, x, m + 1, j$ )
```

- Possiamo estrarre la funzione di ricorrenza dallo pseudocodice

$$T(n) = \begin{cases} 1 & n = 0 \text{ (costo costante se spazio di ricerca è 0)} \\ T(n/2) + 1 & n > 0 \text{ (costo costante + ricerca su 1/2 spazio)} \end{cases}$$

- Soluzione col Master Theorem:

$$\alpha = 0, \beta = 0 \implies T(n) = \Theta(n^0 \log n) = \Theta(\log n)$$